ADVANCE CMOS VLSI

FINAL PROJECT REPORT

Submission Date – May 6th 2016

Submitted by

CHINMAI

CONTENTS

1. INTRODUCTION

This project is designed to apply all the Cadence tools that were covered in the course so far:

- Virtuoso schematic and layout editor
- Verilog Behavioral Modeling
- Nclaunch and Ncsim simulation
- RTL Compiler Synthesis
- Encounter layout synthesis

Using these tools a Floating Point Processor was designed, built and simulated. Then a control was designed to input operations for the calculation of rotation angle. This was achieved by calculating the Sine and Cosine using simplified Taylor expansion.

2. FLOATING POINT ARTHIMETIC

Floating point representation: Half precision floating point was used which means 16 bit represents the number. The FP number will be considered to be always normalized. The value can be represented as:

$$\boldsymbol{A} = (-\boldsymbol{1})^{[\ ]} \times \boldsymbol{2}^{[\ ][\ ][\ ][\ ][\ ]} \times \boldsymbol{1}.[\ ][\ ][\ ][\ ][\ ][\ ][\ ][\ ][\ ][\ ]$$

Example: 1 10011 0111100000 = 23.5

The half-precision binary floating-point exponent is encoded using an offset-binary representation, with the zero offset being 15; also known as exponent bias.

Emin = 000012 − 01111_2 = −14
Emax = 111102 − 01111_2 = 15
Exponent bias = 01111_2 = 15

Floating Point Arithmetic: We are performing floating point arithmetic adhering to the lecture notes provided in class.

Addition: Addition operation used in Floating point processor can be represented using a flow diagram in figure 1.0. We compare the exponents of the numbers. Shift the smaller number to the right until its exponent would match the larger exponent. We then add the two mantissas along with significands. Once we obtain the results we normalize the sum, either by shifting it to right, incrementing exponent or shifting it to left or decrementing the exponent.

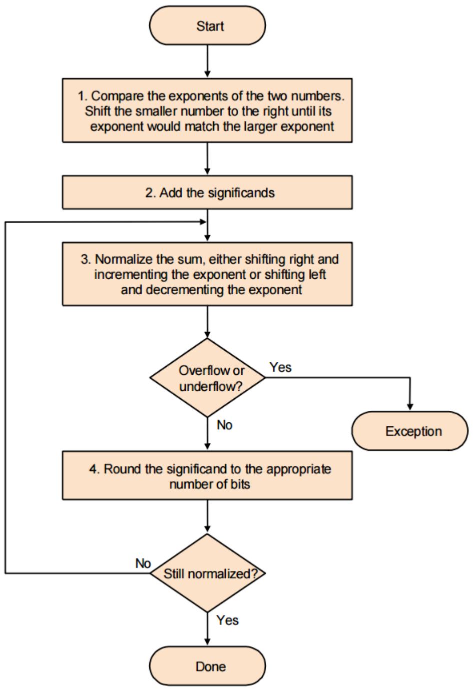

Start
1. Compare the exponents of the two numbers. Shift the smaller number to the right until its exponent would match the larger exponent
2. Add the significands
3. Normalize the sum, either shifting right and incrementing the exponent or shifting left and decrementing the exponent
Overflow or underflow?
Yes
Exception
No
4. Round the significand to the appropriate number of bits
No
Still normalized?
Yes
Done

Figure 1.0

Multiplication:

A multiplication of two floating-point numbers is done in four steps:

• Non-signed multiplication of mantissas: it must take account of the integer part, implicit in normalization. The number of bits of the result is twice the size of the operands
• Normalization of the result: the exponent can be modified accordingly
• Addition of the exponents, taking into account the bias
• Calculation of the sign = xor sign bits of two numbers

3. FLOATING POINT PROCESSOR ARCHITECTURE AND OPCODE SCHEMATIC

To Design a Floating Point Processor (FPP) architecture and opcode scheme. Following are the design constraints considered:

- The processor will have a 16-bit input/output data bus
- 8 bits for opcode
- Four 16-bit registers FP0, FP1, FP2, FP3.
- Every register will use half precision floating point format.

a) OPCODE SCHEME:

Following opcode scheme has been adapted by us in this project:

Function Bits				Source Bits		Destination Bits	
7	6	5	4	3	2	1	0

Function Bits:		Source Bits / Destination Bits:
LOAD	= 4'b0000	FP0 = 00
MOV	= 4'b0001	FP1 = 01
ADD	= 4'b0010	FP2 = 10
NEG	= 4'b0011	FP3 = 11
STORE	= 4'b0100	
MUL	= 4'b0110	
MAX	= 4'b0111	

b) ARCHITECTURE DESIGN:

FPP consists of ALU unit, Register Unit and Controller. We have designed the ALU unit to perform Negation, Load, store, Maximum, Multiply, Addition operations for Half Precision floating point numbers. Figure 1 gives the pictorial representation of the architecture.

Register Unit – Consists of a Data Bus, 4 main registers and opcode. This Unit is connected to ALU unit. On receiving opcode from controller, FPP sends the Function and data present in the register specified by the opcode to ALU and generates pulse called St Flag. This pulse indicates the ALU to start performing respective function. It continues to monitor Enable flag to go high, so that the resultant data can be store back to Destination register or sent to Data Bus depending on the operation.

ALU Unit – Receives function and data from Register Unit. On seeing St Flag go high, ALU starts to perform the processor functions and sets enable low until the operations have completely. Once the operation has finished, it sends an enable signal to Register Unit, indicating this operation has completed.

Number of Cycles taken by the ALU Functions :

Function	Cycles
LOAD	6
STORE	6
MOV	6
MUL	8
ADD	9
NEG	6

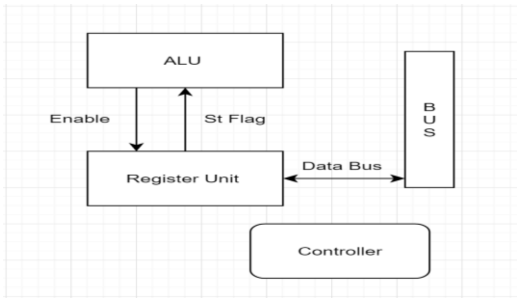


Figure : 1.1 Schematic

Controller Unit - Controller unit sends instructions and DATA. This unit performs operations required to calculate 2D rotated coordinates (x',y') for a point (x,y) which is rotated by θ radians. It calculates Sinɵ and Cosɵ using Taylor series and then calculates rotation angle by following method.

Given Values: X, Y, ϴ

To calculate Sinϴ and Cosϴ

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!}$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!}$$

Rotated Co- ordinates: X' = X*Cosϴ – Y*Sinϴ   Y' = X*Sinϴ + Y*Cosϴ

Controller Code :

```
// calculating Sin theta

LOAD_FP0 theta;          //0.523599 rad
MOVFP0_FP1;
MULFP0_FP1; // theta^2 0_01101_0001100010
MULFP0_FP1; //theta^3 0_01100_0010010110
REGD <= 16'b1_01100_0101010101 ; //-1/3! = -0.1666666
LOAD_FP2;
MULFP1_FP2; // FP2 = theta^3 *(-1/3!) 1_01001_1000011100
MULFP0_FP1;// FP1 = theta^4 0_01011_0011001101
MULFP0_FP1;   //FP1 = theta^5 0_01010_0100000110
REGD <= 16'b0_01000_0001000100 ; // 1/5! = 0.008333
LOAD_FP3;
MULFP1_FP3;//FP3 = theta^5 *(1/5!) 0_00011_0101011011
ADDFP3_FP2;// FP2 = theta^3 *(-1/3!) + theta^5 *(-1/5!)
MULFP0_FP1;// FP1 = theta^6 0_01001_0101000110 (0_01001_0101000010)
MULFP0_FP1;//FP1 = theta^7 0_01000_0110000101 (0_01000_0110000001)
```

```
REGD <= 16'b1_00010_1010000000  ; // -1/7! = -0.00019841269
LOAD_FP3;
MULFP1_FP3;//FP3 = theta^7 *(-1/7!) //00000 ****
ADDFP3_FP2;//FP2 = theta^3 *(-1/3!) + theta^5*(1/5!)+theta^7 *(-1/7!)
MOVFP2_FP1;
ADDFP0_FP1; //0_01110_0000000000//FP1 = sin(theta) FP0 = theta
```

// calculating Cos theta

```
MOVFP0_FP2;
MULFP0_FP2;// FP2 = theta^2 FP0 = theta 0_01101_0001100010
REGD <= 16'b1_01110_0000000000   ; // -1/2! = -0.5 1_01110_0000000000
LOAD_FP3;
MULFP2_FP3;//FP3 = theta^2 *(-1/2!) 1_01100_00011000010
MULFP0_FP2;// FP2 = theta^3 0_01100_0010010110
MULFP0_FP2;// FP2 = theta^4 0_01011_0011001101
REGD <= 16'b0_01010_0101010101   ; // +1/4! = 0.04166666
LOAD_FP0;
MULFP0_FP2; // FP2 = theta^4 *1/4! 0_00110_1001101001
ADDFP2_FP3; //FP3 = theta^2 *(-1/2!) + theta^4 *(1/4!)
REGD <= theta; //
LOAD_FP0;
MOVFP0_FP2;

MULFP0_FP2; // FP2 = theta^2 0_01101_0001100010
MULFP0_FP2; // FP2 = theta^3 0_01100_0010010110
MULFP2_FP2; // FP2 = theta^6 0_01001_0101000001
REGD <= 16'b1_00101_0110110000; //-0.001388888 = -1/6!
LOAD_FP0;
MULFP0_FP2; // FP2 = - theta^6 * 1/6! //0
ADDFP2_FP3; // FP3 = theta^2*(-1/2!)+theta^4*(1/4!)+-theta^6 * 1/6!
REGD <= 16'b0011110000000000; //1
LOAD_FP0;
ADDFP0_FP3; // FP3 = 1 + theta^2 *(-1/2!) + theta^4 *(1/4!) + - theta^6 *  1/6!
MOVFP3_FP2; // FP1 = Sin(theta) FP2 = Cos(theta)
```

// calculating rotation angle x'

```
REGD <= x; //7
LOAD_FP0;
MULFP2_FP0; // FP0 = xcos(theta) 0_10001_1000001111
REGD <= y; //34
LOAD_FP3;
NEGFP3_FP3;
MULFP1_FP3;// FP3 = -ySin(theta) 1_10011_0001000000
ADDFP3_FP0 // FP0 = x' 1_10010_0101111000
STORE_FP0;
```

//calculating y'

```
REGD <= x; //7
LOAD_FP0;
MULFP1_FP0;// FP0 = xSin(theta) 0_10000_1100000000
REGD <= y; //34
LOAD_FP3;
MULFP2_FP3;// FP3 = yCos(theta) 0_10011_1101011001
ADDFP3_FP0;// FP0 = y' 0_01110_1011101011
STORE_FP0;
```

3. RC Synthesis and Chip Design :

The schematics of the register and FPU were synthesized using RTL compiler. Then using the flat file generated, a layout was synthesized in Encounter.

i. Power rings and strips were used
ii. Clock tree was specified
iii. Module placement was adjusted

This generated layout is then streamed into a pad-frame with specified are area.
The I/O pins of the frame get routed with the I/O and power pins of the synthesized layout. The resulting chip is tested in Virtuoso. Table 1.3 depicts the area, power consumed by the chip.

Parameter	**Value**
Worse Case Path	15333 ps
Power	0.18 w
Area	760,716 *sqrµ*
Number of Cells	2295

Table 1.3

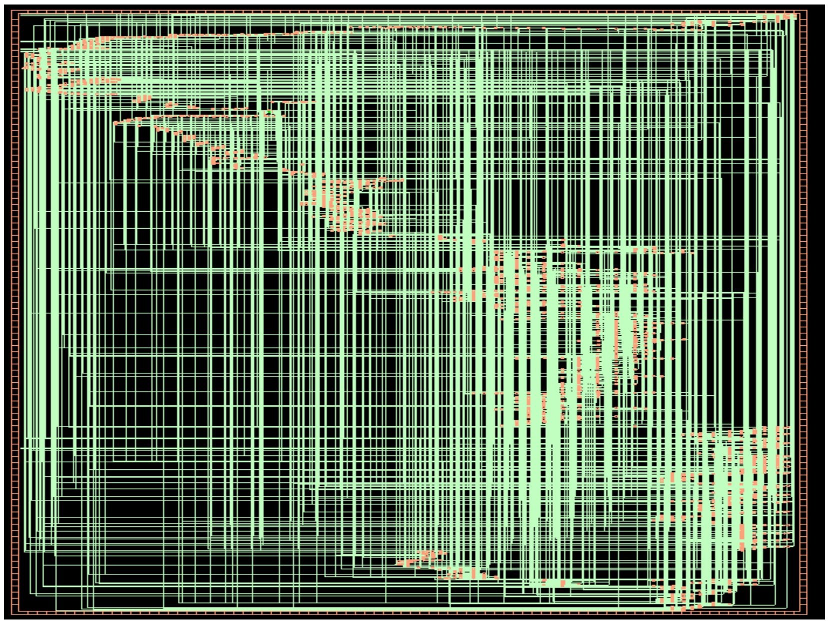


Figure 1.2 RC Synthesis

On designing the layout of the whole unit together, we could not fit the entire layout in the Frame. Therefore a separate layout was designed for Register Unit and ALU Unit.

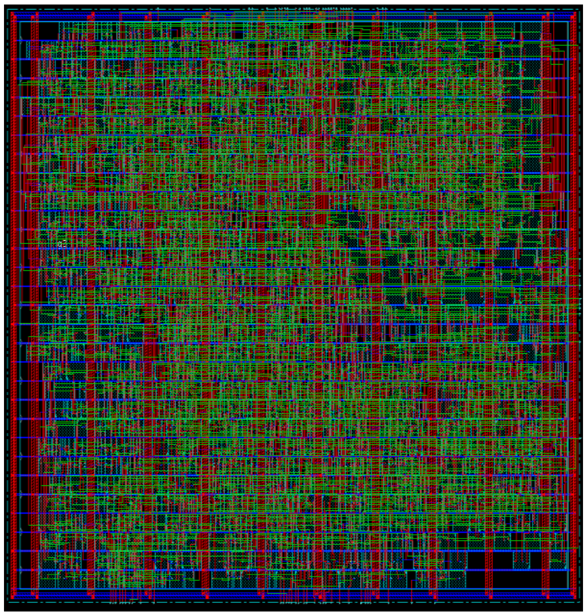


Figure 1.3 Layout of ALU Unit

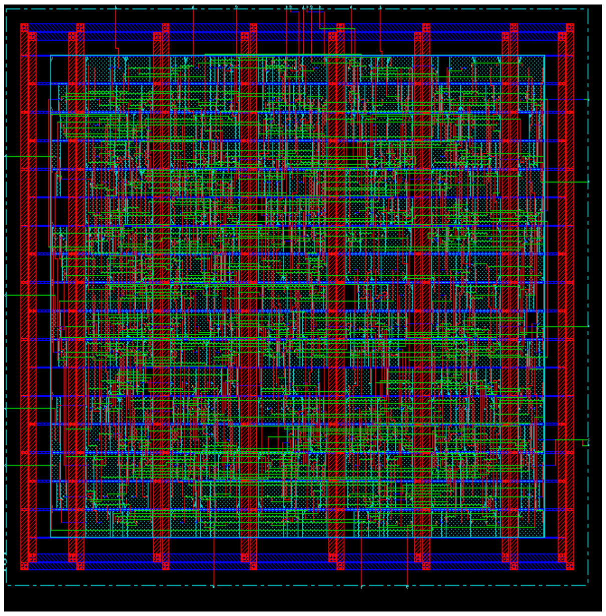
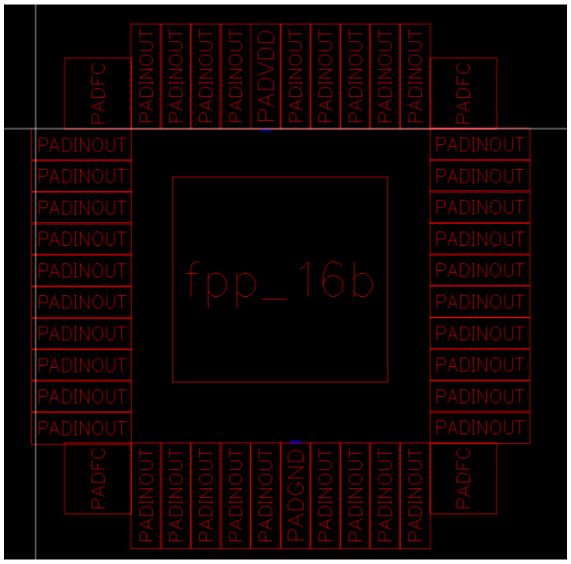


Figure 1.4 Layout of Register Unit



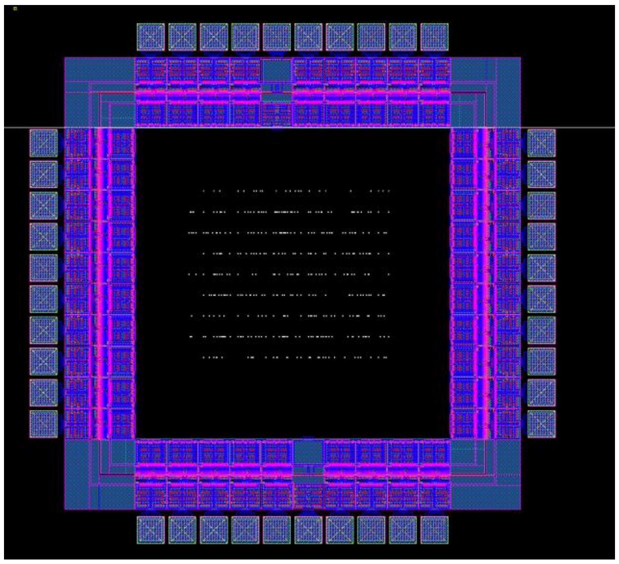


Figure 1.5 Frame Result

4. RESULTS:

This FPP architecture was used to calculated x’ and y’ coordinates of (7,34) for different angles and following were the results obtained by the FPP:

	Expected Result	Result Obtained
Sin(0.523599)	0.5 (0_01110_0000000000)	0.5 (0_01110_0000000000)
Cos(0.523599)	0.8660 (0_01110_1011101101)	0.866 (0_01110_1011101101)
Sin(0.75)	0.68163 (0_01110_0101110011)	0.6821 (0_01110_0101110101)
Cos(0.75)	0.7316 (0_01110_0111011010)	0.73144(0_01110_0111011010)
Sin(1.75)	0.9839 (0_01110_111011111)	0.984375(0_01110_111110000)
Cos(1.75)	-0.17824 (1_01100_0110110100)	-0.1806(1_01100_011100100)
Sin(-0.523599)	-0.500 (1_01110_0000000000)	-0.5 (1_01110_0000000000)
Cos(-0.523599)	0.8660 (0_01110_1011101101)	0.866 (0_01110_1011101101)

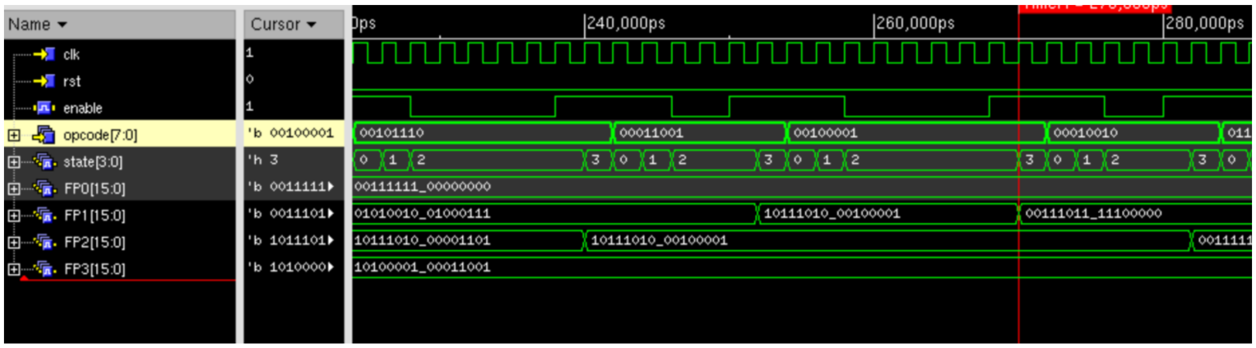
Table 1.1

x = 7 , y = 34

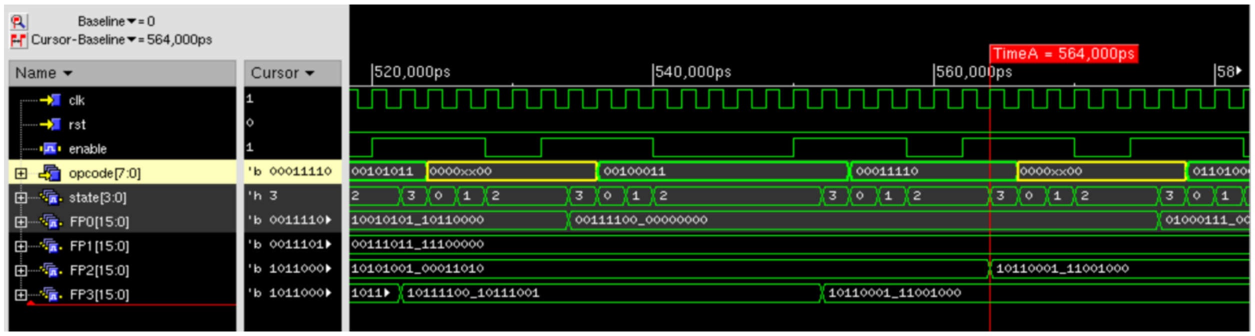
ϴ	Expected X’	RESULT Obtained	Expected Y’	RESULT Obtained
0.523599	-10.93 (1_10010_0101110111)	-10.9375(1_10010_0101111000)	32.94 (0_10100_0000011110)	32.94(0_10100_000001 1110)
0.75	-18.05 (1_10011_001000011)	-18.07 (1_10011_0010000101)	29.64 (0_10011_1101101000)	29.64(0_10011_11011010 00)
1.75	-34.70 (1_10100_0001010110)	-34.08(1_10100 _0001010111	0.833 (0_01110_1010101010)	0.733(0_01110_1000000 0)
-0.523599	23.0621 (0_10011_0111000011)	23.0625(0_10011_0111000100)	25.9448 (0_10011_100111100)	25.9448(0_10011_100111 1100)

Table 1.2

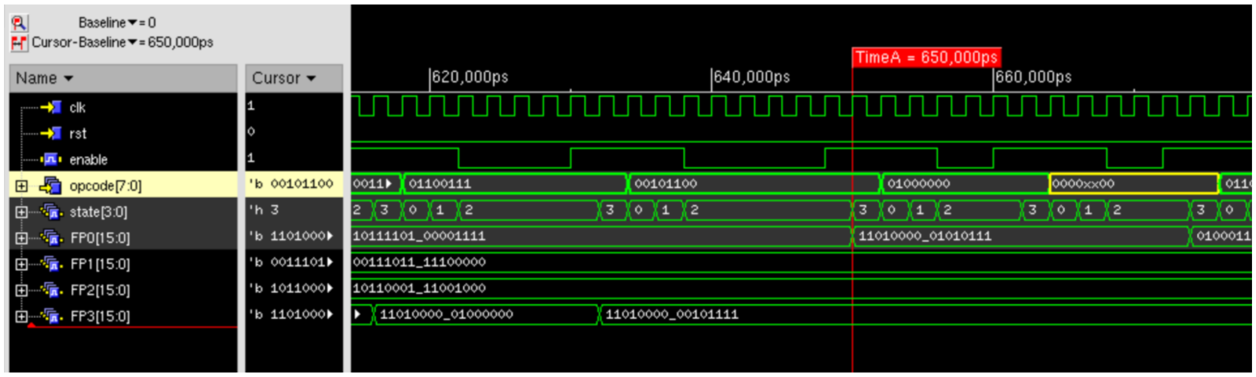
Below are the results obtained for x= 7, y = 34, theta = 1.75 rad



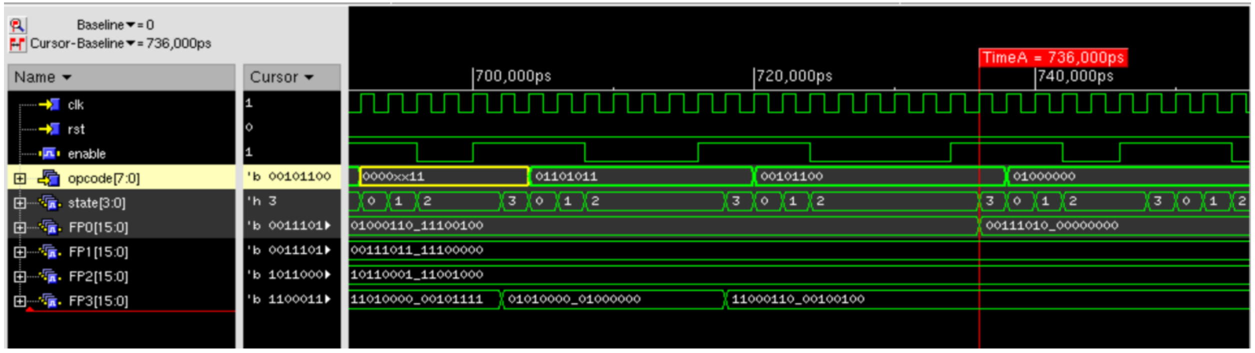
(a) FP1 = Sin(1.75) 0_01110_111110000



(b) FP2 = Cos (1.75) 1_01100_011100100



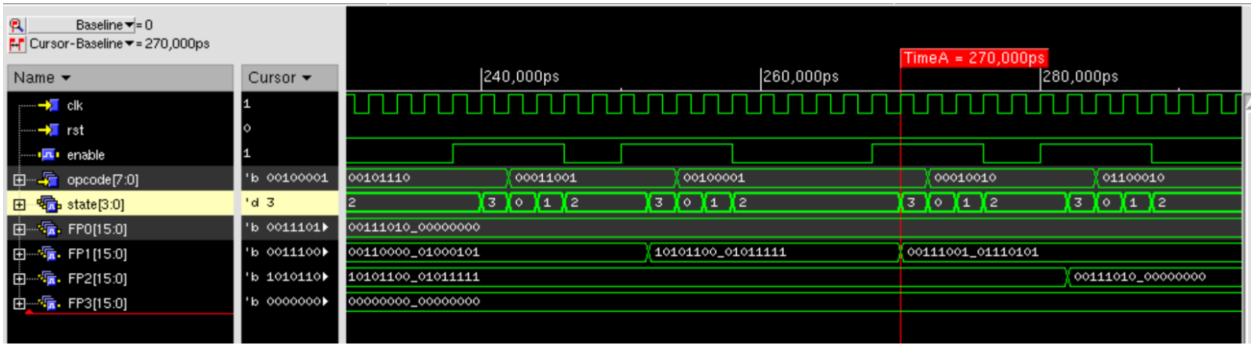
(c) FP0 = X' 1_10100 _0001010111



(d) FP0 = Y' 0_01110_1000000000

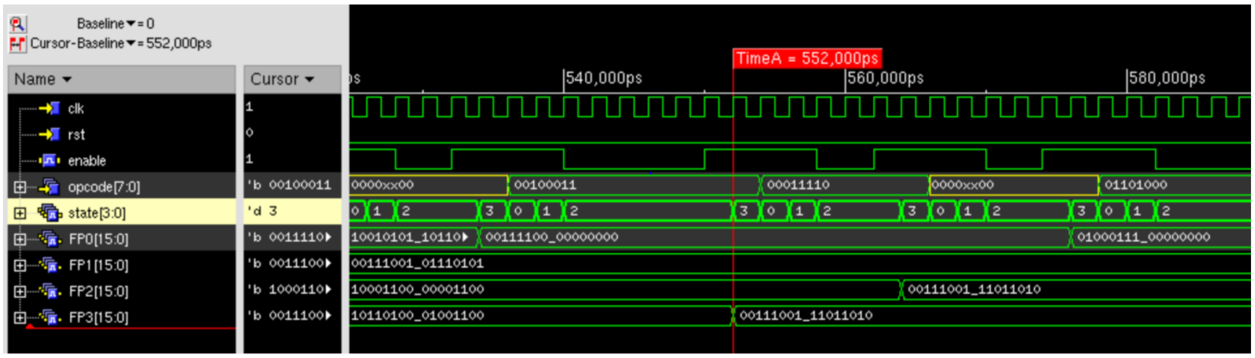
Figure 1.6 Waveforms

Below are the results obtained for x= 7, y = 34, theta = 0.75 rad



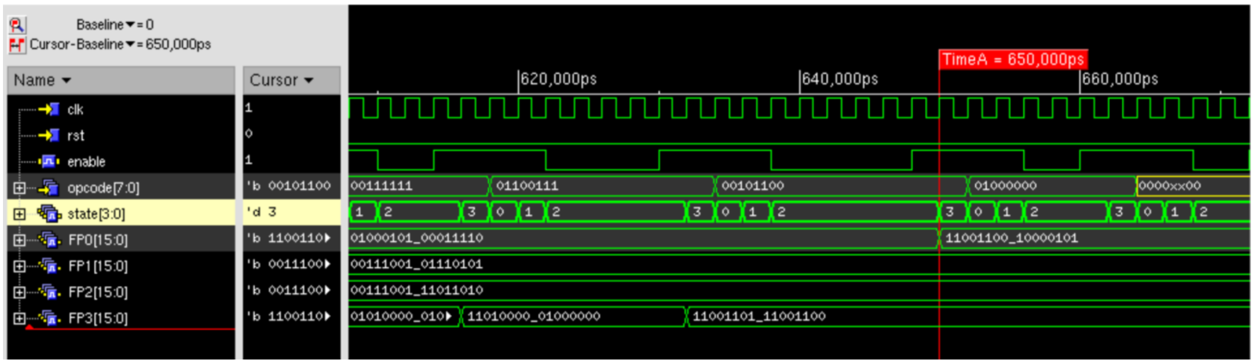
```
Baseline = 0
Cursor-Baseline = 270,000ps
TimeA = 270,000ps      |240,000ps   |260,000ps   |280,000ps
Name           Cursor
clk            1
rst            0
enable         1
opcode[7:0]    'b 00100001   00101110 | 00011001 | 00100001 | 00010010 | 01100010
state[3:0]     'd 3          2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2
FP0[15:0]      'b 0011101▸   00111010_00000000
FP1[15:0]      'b 0011100▸   00110000_01000101 | 10101100_01011111 | 00111001_01110101
FP2[15:0]      'b 1010110▸   10101100_01011111 | 00111010_00000000
FP3[15:0]      'b 0000000▸   00000000_00000000
```

(a) FP1 = Sin(0.75) 0_01110_0101110101



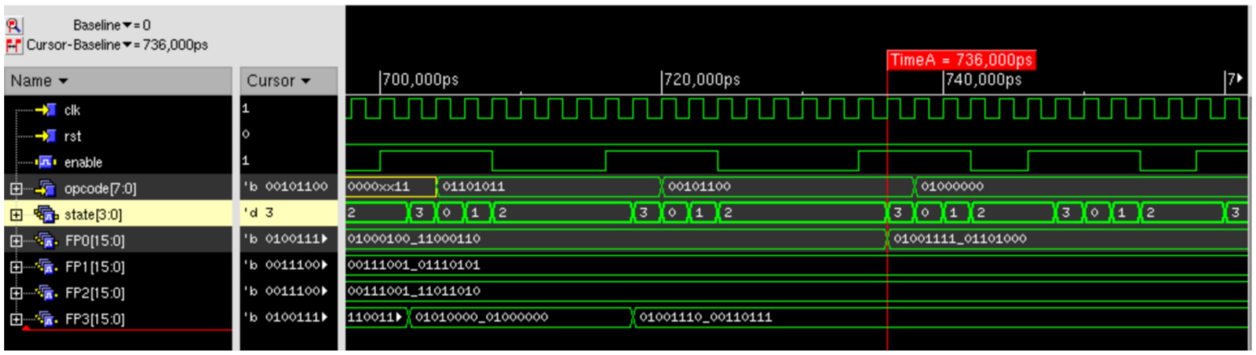
```
Baseline = 0
Cursor-Baseline = 552,000ps
TimeA = 552,000ps      ps   |540,000ps   |560,000ps   |580,000ps
Name           Cursor
clk            1
rst            0
enable         1
opcode[7:0]    'b 00100011   0000xx00 | 00100011 | 00011110 | 0000xx00 | 01101000
state[3:0]     'd 3          0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2
FP0[15:0]      'b 0011110▸   10010101_10110▸ | 00111100_00000000 | 01000111_00000000
FP1[15:0]      'b 0011100▸   00111001_01110101
FP2[15:0]      'b 1000110▸   10001100_00001100 | 00111001_11011010
FP3[15:0]      'b 0011100▸   10110100_01001100 | 00111001_11011010
```

(b) FP2 = Cos (0.75) 0_01110_0111011010



```
Baseline = 0
Cursor-Baseline = 650,000ps
TimeA = 650,000ps      |620,000ps   |640,000ps   |660,000ps
Name           Cursor
clk            1
rst            0
enable         1
opcode[7:0]    'b 00101100   00111111 | 01100111 | 00101100 | 01000000 | 0000xx00
state[3:0]     'd 3          1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2
FP0[15:0]      'b 1100110▸   01000101_00011110 | 11001100_10000101
FP1[15:0]      'b 0011100▸   00111001_01110101
FP2[15:0]      'b 0011100▸   00111001_11011010
FP3[15:0]      'b 1100110▸   01010000_010▸ | 11010000_01000000 | 11001101_11001100
```

(c) FP0 = X’ 1_10011 _0010000101



```
Baseline = 0
Cursor-Baseline = 736,000ps
TimeA = 736,000ps      |700,000ps   |720,000ps   |740,000ps   |7▸
Name           Cursor
clk            1
rst            0
enable         1
opcode[7:0]    'b 00101100   0000xx11 | 01101011 | 00101100 | 01000000
state[3:0]     'd 3          2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3
FP0[15:0]      'b 0100111▸   01000100_11000110 | 01001111_01101000
FP1[15:0]      'b 0011100▸   00111001_01110101
FP2[15:0]      'b 0011100▸   00111001_11011010
FP3[15:0]      'b 0100111▸   110011▸ | 01010000_01000000 | 01001110_00110111
```

(d) FP0 = Y’ 0_10011_1101101000

Figure 1.7 Waveforms

5. Presentation slides and CODE

ECE262Final_project
_presentation (2).ppt

Final Code.zip